Using Redundancy to Improve Robustness of Distributed Mechanism Implementations

Using Redundancy to Improve Robustness of Distributed Mechanism Implementations

Jeffrey Shneidman* David C. Parkes*

ABSTRACT

This paper introduces computation compatibility and communication compatibility as requirements for a distributed mechanism implementation. Just as payments are used to create incentive compatible mechanisms, some technique must be used to create computation/communication compatible mechanisms. This paper explores computation redundancy and communication redundancy as two such techniques. This paper uses interdomain routing as an example domain, and considers where redundancy can succeed and fail in addressing cheating with respect to computation and communication.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity; J.4 [**Computer Applications**]: Social and Behavioral Sciences—*Economics.*

General Terms

Algorithms, Economics.

1. INTRODUCTION

Mechanism design (MD) studies how best to specify the strategic situation, or "rules of the game", so that the system as a whole exhibits good behavior in equilibrium when self-interested nodes pursue self-interested strategies. Classical MD assumes that the players feed their calculated strategies to a special obedient center that performs the mechanism calculation and declares the outcome. This classic approach is not realistic in many computational settings, where networks may consist of strategizing agents and a special center may not exist, with computation and communication spread over several rational nodes. *Distributed* algorithmic mechanism design (DAMD) [2] more accurately models network situations, where both agents and resources are distributed.

One large problem in implementing distributed mechanisms is that mechanism design ideas were originally intended to reason only about agent inputs [4]. It is always assumed in traditional MD literature that messages from agents to the center flow freely and that the center performs all necessary computation. In this paper, we are concerned with equilibrium *message passing* and *computation* behaviors by strategic nodes, when the computational nodes and strategic nodes are the same.

*Division of Engineering and Applied Sciences, 33 Oxford Street, Harvard University, Cambridge, MA 02138. {jeffsh,parkes}@eecs.harvard.edu. Supported in part by NSF grant IIS-0238147.

EC'03, June 9–12, 2003, San Diego, California, USA.
ACM 1-58113-679-X/03/0006.

2. COMPUTATION AND COMMUNICATION COMPATIBILITY

MD often assumes that a player's strategy stops at the point of information revelation—namely, a strategy only concerns a node's input choice into a mechanism. In a network setting, however, an agent may be asked to do more than provide a value. Specifically, in DAMD, an agent may be asked to relay other agent's values or other data, and implement part of a distributed mechanism computation. We explicitly extend the definition of an agent strategy (and type) to define an agent's behavior when asked to pass messages and perform computation as part of a distributed mechanism.

One nice goal in mechanism design is to create mechanisms that are incentive compatible. Payments are used in mechanism design to bring private-value truth revealing into an agent's equilibrium strategy. This paper introduces the analogous ideas of computation compatibility and communication compatibility as two desirable mechanism implementation properties.

Definition. A distributed mechanism is *computation compatible* if every agent chooses to follow a computation specification dictated by the mechanism creator and perform all computation obediently.

Definition. A distributed mechanism is *communication compatible* if every agent chooses to follow a communication specification dictated by the mechanism creator and forward all messages obediently.

Computation compatibility and communication compatibility are intended to address computation and communication manipulation, just as incentive compatibility is intended to address information-revelation manipulation. Unlike incentive compatibility, which takes *truth-revelation* as the "gold standard" of behavior, computation and communication compatibility must be stated with respect to some externally designed specification.

In the absence of appropriate techniques, it is quite likely that an agent will want to deviate from a computation or drop/change messages from others (cheat) in order to better its own outcome [4]. Just as mechanism designers use payments to create incentive compatible systems, one must use *something* to create communication compatible and computation compatible systems.

This paper explores two forms of **redundancy** that might help designers build mechanisms where full message passing and obedient computation behaviors are enacted by strategic nodes. Redundancy is just one of several approaches, and additional ideas are mentioned elsewhere [4].

3. REDUNDANCY AS A TOOL

Redundancy can be used as a tool to build distributed systems that are computation- and communication compatible. Intuitively, redundancy is a device that can promote coordination around the computation and communication specification provided by a mechanism creator. In this work, we assume that redundancy is used to check whether an individual node has deviated, in which case the mechanism is terminated and no outcome is implemented. We assume that all nodes prefer that a mechanism computes some outcome rather than no outcome, and rely on this in order to avoid having to explicitly define methods to punish deviating nodes.[1]

We consider two forms of redundancy. The first is *redundant communication*. Informally, this means that multiple disjoint paths exist between nodes and are used to relay copies of messages (e.g. node inputs or details of mechanism computation). Sending messages down multiple paths in a biconnected network, where there are at least two independent paths from any node to any other node, can ensure that single node deviations can be detected.[2]

The second form of redundancy is *redundant computation*. This idea proposes that nodes be required to perform parallel (hopefully identical) computations for comparison purposes by a third party. In the centralized mechanism case, this can mean designating another node as a redundant center. In the decentralized mechanism case, where computation is spread throughout the network, some set of alternate nodes might be assigned to receive and the same information and perform identical computations.

4. EXAMPLE: INTERDOMAIN ROUTING

It helps to explore the problem of computation- and communication compatibility in the context of a previously stated problem. Feigenbaum et al. [1] (hereafter FPSS) have previously defined a distributed algorithm to implement a BGP-based mechanism for lowest cost interdomain routing. FPSS focus on whether a Vickrey-Clarke-Groves (VCG) mechanism can be implemented without a center and without requiring an unreasonable amount of additional communication or computation, but focus less on the issue of computation- and communication- compatibility.

There are two logical phases in FPSS. In the first phase, a node finds the lowest cost paths to all possible routing destinations. In the second phase, a node executes a pricing calculation that determines transit node payments (how much it must pay each transit node for traffic originating at it and traveling to the various destinations.) In FPSS, a complete message to a neighbor (and thus the totality of what can be manipulated) consists of three items: a routing table which consists of paths and total path costs, a list of the declared costs of transit nodes, and a sparse pricing table.

A rational node will cheat to achieve at least one of two goals. First, a node can try to increase the amount of money it is paid. Second, a node can try to decrease the amount of money is must pay. We consider each phase in turn.

Phase I. A node broadcasts paths and forms LCPs. Since FPSS assume a static biconnected environment in which no nodes enter or leave the system, and there are no articulation points, it can be seen that any misreporting of a neighbor cost will reset the algorithm as inconsistent information prevents network quiescence. In the FPSS model, inconsistencies are detected through redundant communication since the network is biconnected. Furthermore, it is irrational for a node to drop a message since doing so can only make that node appear less desirable to others. Thus, the first logical phase is communication compatible. Phase I also contains computation, but again, we can show that it is computation compatible.

Phase II. Once the LCPs are calculated, a node is responsible for calculating the payments that it owes to other nodes in the network, and can try to change the amount of money it must pay. Because of the manner in which Vickrey payments are calculated, it is already the case that a node's computation cannot change the amount of money that it is paid by other nodes.

In this logical phase, no communication related to the pricing calculation takes place - a node simply calculates prices that it must pay to others. The problem in Phase II is one of computation compatibility, and redundant computation can help. By simultaneously requiring one or more nodes to perform the pricing computation on behalf of the original node, the rational behavior for an individual node can made compatible with the specified computation. The mechanism is simply terminated whenever the computed payment by the original node, and its checker, does not match.

As an alternative, FPSS suggest *cryptographic signing* of messages, as one way to detect when nodes deviate from their specified computation. All of the messages in Phase I are signed, and then a payment-receiving node that suspects cheating can verify that a pricing calculation was performed correctly. In comparison, the technique of redundant computation is more distributed and more passive, in that a node doesn't need to suspect misbehavior before cheating will be caught. Node pricing calculations are constantly checked, and this load is shared throughout the network. The main challenge is to select checking nodes, and to get the required information to checking nodes without unreasonable communication cost. We leave the complete details of redundant computation within the context of interdomain routing to our longer paper [5]. We demonstrate a solution that combines redundant computation with a small amount of cryptographic signing, just to get the required information to checking nodes.

[1] More sophisticated punishment strategies will be necessary in practice, to prevent the loss of economic efficiency that would arise, for example due to one irrational or faulty node.

[2] An example of this redundant communication technique occurs in previous work on *Distributed Games* [3]. That work explores a *centralized* auction mechanism that executes in a ring-topology network of rational nodes. In this model, it is irrational for a node to drop a message, as this behavior can be detected by the center upon receiving a message via the alternate path. In addition, when redundancy is combined with other information-theoretic tools it becomes irrational for a node to change a message, as this change will not better the outcome of the deviant node [3].

5. REFERENCES

[1] J. Feigenbaum, C. Papadimitriou, R. Sami, and S. Shenker. A BGP-Based Mechanism for Lowest-Cost Routing. In *Proc. of the 2002 ACM Symp. on Principles of Distributed Computing*, pages 173–182, 2002.

[2] J. Feigenbaum and S. Shenker. Distributed Algorithmic Mechanism Design: Recent Results and Future Directions. In *Proc. of the 6th Int. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 1–13, 2002.

[3] D. Monderer and M. Tennenholtz. Distributed Games: From Mechanisms to Protocols. In *Proc. 16th Nat. Conf. on Artificial Intelligence (AAAI-99)*, pages 32–37, July 1999.

[4] J. Shneidman and D. C. Parkes. Rationality and Self-Interest in Peer to Peer Networks. In *Proc. 2nd Int. Workshop on Peer-to-Peer Systems (IPTPS'03)*, 2003.

[5] J. Shneidman and D. C. Parkes. Using Redundancy to Improve Robustness of Distributed Mechanism Implementations. In *Proc. Fourth ACM Conf. on Electronic Commerce (EC'03)*, 2003. Shorter version. Extended version available at http://www.eecs.harvard.edu/~parkes/pubs/redundancy.pdf.